

The REBOOT Approach to Software Reuse

(to appear in Journal of Systems and Software, 1995)

Guttorm Sindre and Reidar Conradi

Norwegian Institute of Technology (NTH), University of Trondheim, Norway

Even-André Karlsson

Q-Labs (Quality Laboratories Sweden AB), Lund, Sweden

Although some companies have been successful in software reuse, many research projects on reuse have had little industrial penetration. Often the proposed technology has been too ambitious or exotic, or did not scale up. REBOOT emphasizes industrial applicability of the proposed technology in a holistic perspective: a validated method through a Methodology Handbook, a stabilized tool set around a reuse library, a training package, and initial software repositories of reusable components extracted from company-specific projects. This article presents the REBOOT approach to software reuse, covering both organizational and technical aspects and the experiences made so far from the applications.

1 Introduction

1.1 Context and motivation

For a long time, reuse [2, 20, 27] has been promoted as a promising approach to improve quality and productivity in software engineering, and some software producing organizations have indeed succeeded in making reuse pay off. Still, reuse has not revolutionized the software engineering industry the way many had hoped for. This is partly because expectations have been too high [6], but also because systematic reuse has not yet been taken up by the bulk of software engineering industry.

The reason for the slow industrial penetration of reuse is that it is a complex issue, inhibited both by organizational and technical barriers [15, 28]. Previous research projects to provide support for reuse have often suffered from the following weaknesses:

- too much emphasis on technical aspects, underestimating the organizational problems, and
- too sophisticated development methods, requiring dramatic changes in practice for software producers.

The motivation of the REBOOT¹ project was to overcome these difficulties by

- paying attention both to organizational and technical aspects,
- proposing a reuse method which could be integrated with existing methods, and
- generally going for *technology consolidation* rather than innovation.

To ensure industrial penetration of the ideas, the project had a strong commitment towards applications (25% of the effort).

1.2 Overview of the problem area and the REBOOT offer

The following problems have been addressed within REBOOT:

- organizational aspects of reuse, concerning
 - *reuse introduction*, i.e., introducing systematic reuse in an organization with little or no reuse experience.
 - *organizational and managerial issues*, i.e., how to organize to get the most out of reuse, and how to manage development projects where reuse is considered.
 - *economic and legal aspects*, which are particularly problematic for inter-company reuse, such as payment for components and liability in case of component failure.
- technical aspects of reuse, concerning
 - *development FOR reuse*, i.e., how to make reusable components.
 - *development WITH reuse*, i.e., how to make use of reusable components in application development.

¹ ESPRIT-III project # 7808 REBOOT (Reuse Based on Object-Oriented Techniques) started September 1990 and had a duration of 4 years. The partners were Bull S.A. (coordinator, France), Cap Gemini Innovation (France), Sema Group SAe. (Spain), Siemens A.G. (Germany), Q-Labs (Sweden), E-P Frameworks (Sweden), TXT Ingegneria Informatica (Italy), and SINTEF/NTH (Norway). The total effort was 124 man-years.

The REBOOT offer for these problems consists of

- a methodology handbook,
- a training package, and
- a prototype reuse support environment.

The *REBOOT Methodology Handbook* [24] provides methodology both for organizational and technical aspects of reuse. The following chapters are included:

1. **Reuse Organization.** This chapter presents various ways to organize for reuse, and also discusses the economic and legal issues involved.
2. **Project Management.** This chapter discusses how to manage projects where software is developed FOR and/or WITH reuse.
3. **Development FOR and WITH reuse.** This chapter provides software engineering guidelines related to the development of reusable components, and the development of applications by means of such components. Some of the guidelines are general, but many presuppose that an object-oriented development method is used.
4. **Library organization.** This chapter presents methodology for organizing and maintaining libraries of reusable components.
5. **Metrics.** This chapter provides process metrics for evaluating reuse activities, and product metrics to evaluate the quality and reusability of components.
6. **Reuse introduction.** This chapter discusses how to introduce reuse in an organization.

The *Training Package* consists of three courses (slides and scripts):

- **Managerial Briefing.** This is a two hour briefing for managers, outlining the possible benefits of reuse, as well as the most important problems. The mission of this course is to increase managers' awareness of reuse and convince them that it is worth considering.
- **Reuse Beginners Course.** This is a course for developers who have little knowledge about software reuse principles.
- **Advanced Reuse Course.** This is a course for more advanced developers, who have already been through the Beginners Course, or otherwise acquired knowledge about software reuse.

The REBOOT environment gives tool support for the technical aspects of the methodology. To have maximum possibility for industrial penetration, REBOOT does not require a shift in development method (or tool) but rather offers a potential add-on to deal with the reuse-specific activities. This is indicated in Figure 1. Hence, the organization can keep on using the same CASE-tool and development method as before, just with some integration with the REBOOT method. REBOOT comes in to deal with the activities to get components in and out of the library, whereas ordinary development can still be handled by the existing approach. Several integrations with existing methods and tools have already been performed.

The rest of the article is structured as follows: Section 2 discusses the organizational part of the REBOOT methodology, and section 3 the technical part. Section 4 summarizes the experiences made from the applications. Section 5 makes a comparison with related work. Finally, section 6 concludes the article.

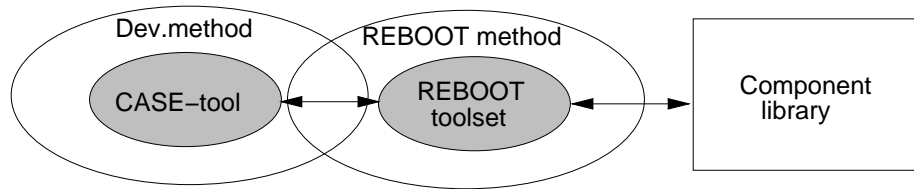


Figure 1: The general idea of the REBOOT tool-set

2 Organizational aspects

2.1 Reuse introduction

Several researchers and practitioners have pointed out problems in the reuse introduction process, e.g., [15, 28]. These problems are based on many factors:

- high initial costs, since there must be investment in 'development FOR' before gains in 'development WITH'
- lack of sound management commitment
- developer skepticism, the 'not invented here'-syndrome
- lack of training for reuse
- low software engineering capabilities

For a reuse initiative to get started, the management must be convinced of possible gains. For this purpose, one of the courses in the REBOOT training package is a two hour managerial briefing. This briefing informs top level managers about the potential benefits of reuse, but also about the problems — to avoid unrealistic expectations.

Reuse introduction is covered by a separate chapter in the REBOOT Methodology Handbook. An overview of the REBOOT reuse introduction process is shown in Figure 2. The left column shows the activities to be undertaken, the middle column the roles involved in these activities, and the right column the models supporting the activities. During initiation, the objectives for the reuse initiative are determined, such as reduced costs or time-to-market, improved quality and productivity. In the second stage, the organization's reuse opportunities are analysed, and its reuse maturity is assessed.

The assessment of the organization's maturity is based upon REBOOT's *reuse maturity model*, inspired by SEI's CMM [36]. The same five maturity levels are used, i.e., chaotic, repeatable, defined, managed, and optimized, but the assessment is specifically related to reuse capabilities, not the overall capabilities for software production. Hence, five *key reuse areas* have been defined according to the five parts of the REBOOT methodology², and for each of these several *key reuse factors* have been identified, as shown in Table 1. Each of these factors can be mastered to various degrees by the organization, thus determining its maturity level. The assessment will discourage organizations from adopting technology they are not yet mature for, and it will indicate what steps to take next to improve the reuse capabilities.

The implementation phase of Figure 2 will have to include training. In addition to the already mentioned managerial briefing, the REBOOT training package contains courses for developers,

²Cf. ch. 1–5 of the Methodology Handbook. The topic of ch. 6, reuse introduction, is something which must be done before any reuse can take place, and is consequently not considered a key reuse area.

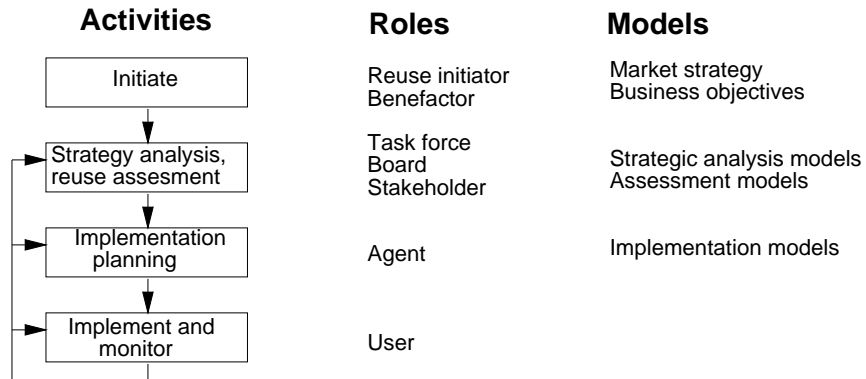


Figure 2: The reuse introduction process

Organization	Project management	Development process		Library	Metrics
		FOR reuse	WITH reuse		
Reuse strategy	External coordination	Development process integration		Component information	Product metrics
Reuse assessment	Project planning	Type of produced/reused information		Component classification	Process metrics
Legal issues	Project tracking	Variability analysis	Functionality evaluation	Change management	
Cost & pricing	Staffing	Generality expression	Reuse cost evaluation	Library maintenance	
Product offer		Cost/benefit analysis	Adaptation/integration		

Table 1: The REBOOT key reuse areas and factors

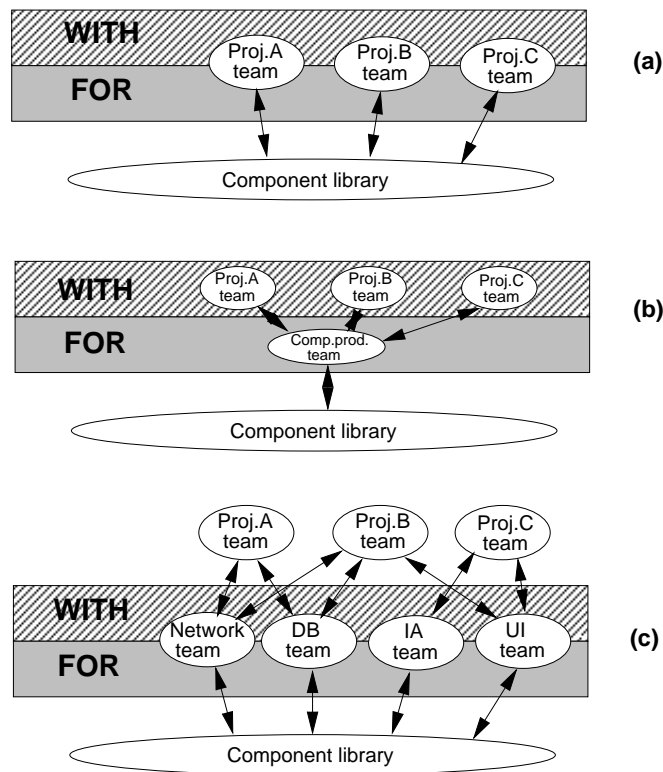


Figure 3: Reuse organizations: project-oriented (a), component-production-oriented (b), and domain-oriented (c).

both at beginners and advanced levels. In addition to increasing the organization's software engineering capabilities, these courses can help to reduce developer skepticism towards reuse.

2.2 Organizational and managerial issues

Even after reuse has been introduced in an organization, there are important issues to deal with:

- how to organize to get the most out of reuse, and
- how to manage projects developing FOR and/or WITH reuse.

Within REBOOT, several ways to organize for reuse have been discussed in the Methodology Handbook. The reuse organization can be structured around *project*, *component-production*, or *domain*, as indicated in Figure 3 (a), (b), and (c), respectively.

All these organization models have their pros and cons. The project-oriented organization has the least overhead connected to reuse, but it is a limited solution because there is no team directly responsible for maintaining the component library, and a lack of inter-team communication will imply that some reuse opportunities may be missed. In the component-production approach, one specific team is responsible for the production of reusable components and the maintenance of the library. This gives good control over the library, but the component-production team will need to have a broad competence, being able to produce any kind of component. More specialization is achieved by the domain-oriented approach, where separate teams have the responsibility for the production of different kinds of components. This yields a secure way of producing reusable components, increasing the chances for a correct identification of reuse needs. The disadvantage of

this approach is the increased communication overhead, and the organization must be larger than some critical size for this model to be possible.

In addition to discussing these organization models, the Methodology Handbook provides more detailed guidelines on how to manage a project developing software FOR and/or WITH reuse, pointing out risks involved in such projects and identifying various roles, such as reuser, component provider, integrator, adaptor, maintainer, project manager, product manager, and product line manager. The possible role conflicts between these are analysed, and guidelines are provided for project management to minimize risks and deal constructively with the role conflicts.

The metrics chapter provides process metrics for evaluating the cost and benefit of reuse activities. These evaluations can provide interesting feedback to guide further improvements in the organization and development process.

3 Technical aspects

3.1 Tool-set overview

When it comes to the technical solutions for reuse, REBOOT takes a component library approach, where reuse is done by retrieving reusable components from the library and integrating them in new applications. Although the component library approach has sometimes been unfavourably compared to transformational approaches [27], it seemed to be the best choice for REBOOT since it does not demand any change of development method or language by the mainstream software producer.³ As stated before, REBOOT's ambition was never to create a new development method and CASE tool, only to support the activities directly related to reuse, such as

- development FOR reuse
 - *re-engineering*: obtaining reusable components from existing, application-specific artifacts — typically old code.
 - *qualification*: ensuring that components to be entered into the library are of acceptable quality. In addition to measuring quality, the REBOOT qualification service also collects information about the component, such as development history and context of use, which can be useful for subsequent evaluation (below).
 - *classification*: classifying the component to facilitate subsequent retrieval. In repositories with a reasonably small number of components this may not be necessary, but if the number of components gets large, it may be useful.
- development WITH reuse
 - *retrieval*: finding components according to some need in the application being developed.
 - *evaluation*: evaluating the quality and appropriateness of candidate components (after retrieval).
 - *adaptation*: adapting the component if it cannot be reused quite as-is. For instance, it may require specialization or name changes to fit into the application.

Ordinary development from scratch — of applications or of reusable components — is not supported by REBOOT. This is assumed to be covered by existing methods and CASE tools. REBOOT

³A preference for OO development methods was made, though, but this is anyway a popular trend in industry today. Besides, many parts of the REBOOT technology, and all the methodology for the organizational aspects, will be applicable also for non-OO development methods.

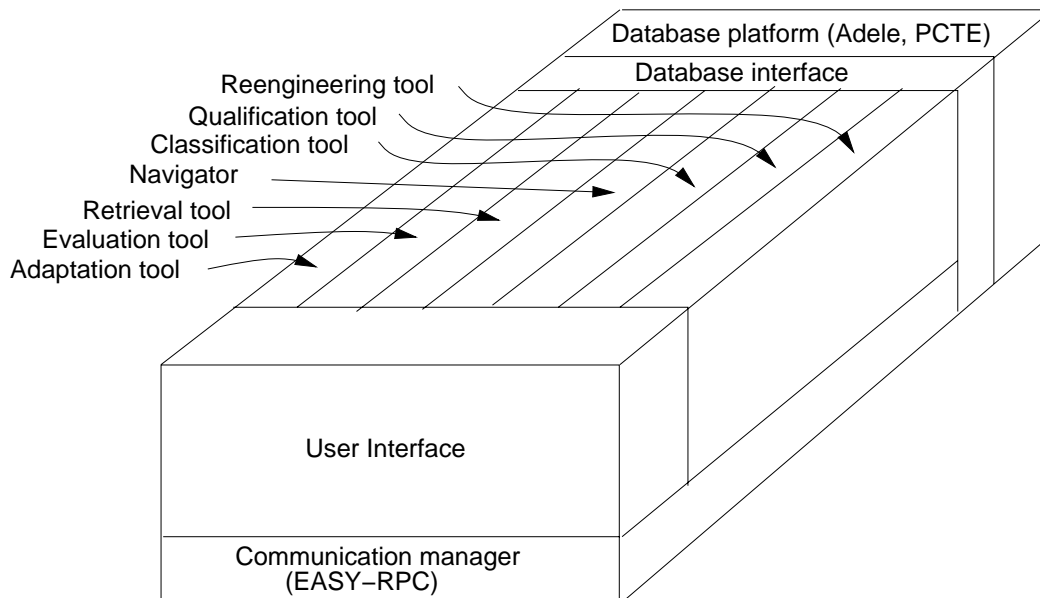


Figure 4: The REBOOT reuse support environment

provides a set of tools for the activities listed above, with a common user interface and database support, as indicated in Figure 4. SEMA has already made an integration between the REBOOT environment and their own development tool Concerto⁴, thus obtaining a CASE tool with reuse support. Similarly, TXT has made an integration with Dragoon⁵. When it comes to methods, REBOOT adaptations have been made for several mainstream approaches for object-oriented analysis and design (e.g., [3, 8, 18, 40]), as well as the Cleanroom approach [32].

REBOOT supports reuse in all phases of development, so components can include analysis models, designs, code, documentation, and test suites. Some of the support assumes that components are object-oriented. For designs, two main kinds of components are supported: classes [7, 39] and design frameworks [22]. The Methodology Handbook provides guidelines on how to develop reusable classes and frameworks. Details on the environment have been presented in [13].

In the following, the support for some of the activities listed above will be discussed more thoroughly, noting initially that they are related in pairs: re-engineering and adaptation, qualification and evaluation, classification and retrieval.

3.2 Re-Engineering and adaptation

The closely related themes of *reverse engineering* (analysing existing code to recover design) and *re-engineering* (transforming existing code to meet new requirements) [30] attract significant research interest nowadays, as indicated by [20, 47]. For many companies it does not feel tempting to start a reuse initiative unless they can reuse parts of previously developed applications, thus reducing the high initial costs for development FOR reuse. However, previously developed systems often tend to have a low reusability.

⁴Concerto is a CASE tool by SEMA Group SAe., specifically intended for the development of large, technical applications. It is recommended by ESA/ESTEC for space projects. It is available on all major workstations, and is currently in use on more than 400 stations throughout Europe.

⁵Dragoon is a programming support environment developed in the ESPRIT project DRAGON to support reusability for Ada. It consists of a preprocessor to translate an extended OO Ada notation into pure Ada code, as well as a library manager and browser.

To deal with this problem REBOOT supports *re-engineering for reuse*, whose purpose is somewhat different from re-engineering in general. Re-engineering usually means to improve an entire application. The REBOOT approach, on the other hand, leaves the system itself unchanged, only extracting reusable components from it. Hence, it is possible to concentrate only on those parts of the system that have a high potential for reuse, which makes the task somewhat simpler.

The basic idea is to analyse existing, usually non-OO, source code in order to identify parts with a strong inner coupling, which can be candidates for reusable objects. An overview of the re-engineering method is shown in Figure 5. The method has mainly been developed by Siemens and is partly automated. First, source code analysis obtains the necessary cross-reference information. Since the information created by commercially available parsers is too limited, special purpose parsers have been developed within the REBOOT project (Ada, C, C++). Notice that whereas some of the tool support — such as parsing — must necessarily deal with concrete programming languages, the re-engineering *method* is mostly language independent.

After source code analysis, some obvious candidates for objects may have emerged, e.g., because they have the names of important objects in the domain. The identification of obvious objects must be done manually by the re-engineer and requires good knowledge about the domain. Inspection of the software's documentation may also be helpful here.

Next, cluster techniques are used to find data items which belong conceptually together and may be assumed to be the private data items of the candidate reusable objects. The clustering is based on a measure of conceptual distance between data items and can be computed in various ways, based on data flow, cross-references, or control-flow branches. This step is performed automatically. A good design should yield a rather balanced cluster-tree, whereas a bad design tends to give a degenerated tree with one single main branch. The cluster-tree is an important input to the next step of identifying the reuse potential. Based on the appearance of the cluster-tree the re-engineer manually determines whether the code has sufficient potential for harvesting of reusable components. If it is decided worthwhile to continue, clustering is now applied also to the functions, attempting to group these functions around their relevant data clusters. Again, similarity computations can be used, but the re-engineer may have to state manually the relative importance of various data to particular functions.

If it is possible to cluster functions around data, such clusters can be taken as objects. Then, the re-engineer will proceed manually, trying to identify the meaning and purpose of these objects. Here, it may be discovered that some more data items should be added to the object, in which case the clustering of data-items must be reiterated. For a functional closure, it may also be necessary to add more functions to the object — typically functions not found by clustering because they do not access the data structures. When a functional closure has been obtained, it is attempted to isolate the object from the environment by replacing accesses to the internal data and providing access routines. Finally, an object-oriented wrapper may be applied to provide an OO interface around the component, while still maintaining the component itself in the old language. These two final steps are automated.

Whereas REBOOT's re-engineering facility is rather advanced, the support for adaptation is simple, merely providing notification in case of name clashes for classes and methods. Hence, a more detailed discussion of the adaptation tool is not considered worthwhile here.

3.3 Qualification and evaluation

Qualification and evaluation both rely on a similar set of product metrics. Reusability is computed from a factor-criteria-metrics model with the following four factors:

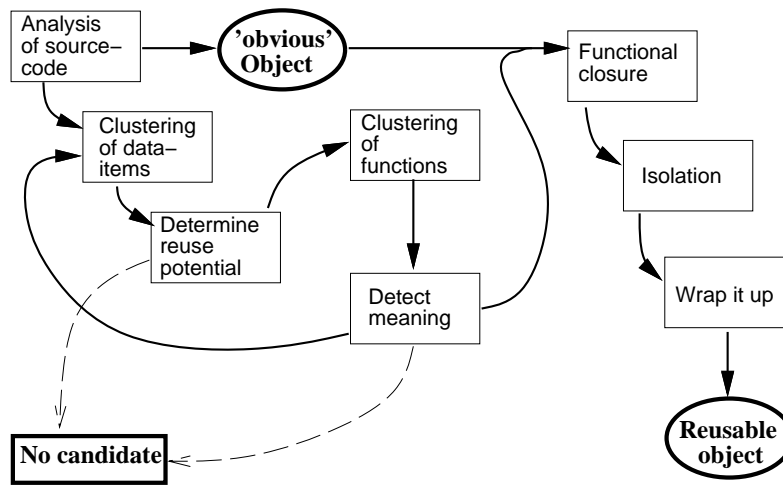


Figure 5: The REBOOT re-engineering method

- **Portability:** The ease with which the component can be transferred from one computer system or environment to another [42].
- **Flexibility:** the ease with which the component can be adapted or modified to different functional needs, or used in different contexts. Flexibility is also referred to as “generality” or “useful complexity” [17], i.e., the functionality offered must be useful and well-defined — otherwise flexibility will only make more complex components without increasing their reusability.
- **Understandability:** A software product is understandable to the extent that its purpose is clear to the prospective reuser [17]. If the component can be reused as-is, understandability is only essential for its interface. If it has to be modified it is also important that the implementation is understandable.
- **Confidence:** The (subjective) probability that a module, program or system performs its defined purpose satisfactorily (without failure) over a specified time in another environment than it was originally constructed and/or certified for. The confidence measure is closely related to reliability, but not exactly the same. Reliability means the probability that the component will function correctly in its current context. Confidence tries to estimate also the probability that it will function correctly when reused in other contexts.

Both a-priori and a-posteriori estimates are provided for these factors. The a-priori estimates are necessary to prevent components of inferior quality from entering the library, since there is no reuse experience for the component at this time. The a-posteriori estimates are based on experience from actual reuses. For each factor, the overall estimate will be a weighted average of a-priori and a-posteriori values, where the latter will gradually be more dominant as the number of experiences increase. The REBOOT product metrics approach has been discussed in more detail in [11].

Attached to each component is also a *reuse history* attribute which can be useful when evaluating its reusability. Here, reusers may describe in free text any problems — or positive experiences — they had reusing the component, and hint on useful adaptations. In some cases, this may be more valuable information than mere numbers for the various factors or criteria.

3.4 Classification and retrieval

Classification is not essential to retrieve components from the REBOOT library — it is also possible to use the navigator for this purpose, browsing the library instead of searching in the associative manner. However, if the number of components in the library grows large, classification will help to get a better overview. Exactly how large *large* is, is hard to say. The Norwegian company TASKON A/S has managed to live for years with a Smalltalk library of approximately 1500 classes, without any systematic classification and retrieval support. However, whereas old employees may easily manage their way around the library, it has been noticed that newly employed have problems, i.e., the 'learning time' for this unstructured library is getting higher as it keeps on growing. Hence, the company itself has signalled interest for classification and retrieval support along the lines of REBOOT, to see if this could make the library more comprehensible.

REBOOT uses a faceted classification scheme [38] with the following four facets:

Abstraction: Usually an OO component can be characterized by a noun, e.g., calendar, flight manager, fire alarm system.

Operations: Components have operations, and these are characterized in the Operations facet.

Operates On: This facet describes the objects that the component acts on, e.g., integers, set, list, resource.

Dependencies: These are non-functional dependencies and characteristics which limit the scope of reuse to a certain context, e.g., C++ code, Unix-based software, HOOD design models.

An example could be a stack (abstraction) of integer (operates on) written in C++ (dependencies) with operations push, pop, top, and swap. It is possible to have more than one term for each facet.

In addition to the terms filled in for the facets, a component will have other attributes, such as size, developer, and date. The distinction between facets and other attributes is that the facets are the properties most relevant for reuse. For the facets, structured term spaces are provided, facilitating fuzzy retrieval. This is not done for ordinary attributes.

Potential reusers often do not know exactly what they are looking for. Moreover, there is a possibility for terminology mismatch between the reuser and library. Consequently, it is useful to be able to retrieve not only components exactly matching the search criteria, but also components which are close to a match. To support this kind of fuzzy retrieval, REBOOT uses *structured term spaces* for the facets. In a term space, all the possible terms for a facet are related in a graph of weighted relations. The REBOOT approach to weighted term spaces is described in [25]. Since the structure of the term space will be domain specific, one must expect to maintain it continuously, according to feedback from the clients of the library (both providers and reusers). REBOOT's proposed heuristics for maintaining a term space are discussed in [41].

4 Applications and experiences

Within REBOOT 25% of the effort has been dedicated to the support and evaluation of reuse applications. Applications have been conducted internally in the partner organizations as well as with external customers, and have provided important feedback for the reuse methodology.

This section will give an overview of the types of applications conducted, and summarize the lessons learnt. A first, overall experience was that it turned out to be much more difficult than expected to initiate and maintain the applications. Some reasons for this are:

- It requires much effort to convince the management to really commit themselves to a reuse initiative.
- Developing FOR and WITH reuse requires training or a relatively large amount of active support.
- The methodology and tools promoted in the beginning of the project were too advanced and were reluctantly received by the pilot applications.

The types of applications conducted are:

- **Product-focussed applications**, supporting projects using object-oriented techniques to develop reusable components, in particular design frameworks for a product family. These projects have been conducted with heavy support from the REBOOT core team providing the ‘development FOR reuse’ expertise. These applications have generally been successful, yielding a lot of practical reuse experience, generalized in a set of concrete object-oriented reuse development guidelines which are included in the REBOOT Methodology Handbook. The organizations involved here have been fairly small, and have not had a large product or process legacy.
- **Process- and organization-focussed applications**, helping organizations to introduce reuse into their existing development process and organizational structure. This work has led to the thorough understanding of how reuse can be incorporated into organizations with a large process and product legacy, organizations which cannot afford to change their product structure or development process dramatically. This has resulted in guidelines for how to adapt development processes and organizations to incorporate reuse.
- **Re-engineering-focussed applications**, helping organizations to extract reusable components from their existing products. Large amounts of money have been invested in old products, but they are usually written in non-OO languages and do not have the structure to be reused. These applications have provided requirements for the re-engineering method which helps to identify potentially reusable components in source code. The re-engineering method has been tried on several C and assembly applications with a size up to 30 KLOC. The results have been really promising, illustrating that the method works for real-life software. However, it should be noted that the possibility for harvesting reusable components depends highly on the quality of the design — in some of the applications investigated, the design was so bad that no candidate components could be obtained.
- **Project management applications**, assisting project managers with assessing their project’s reuse potential. It has hence been the responsibility of the REBOOT consultants to ensure that these reuse opportunities have been utilized. This has provided feedback on the importance of planning the reuse activities with a specific goal and budget, and to follow them up like all other activities. In one of these projects, surprisingly, the investment in development FOR reuse was recaptured already within the first cycle, i.e., *without a single reuse yet being made*. This was because the extra resources for making a component more general for future projects also turned out to make the component more maintainable as requirements were changing during the project.
- **Strategic reuse applications**, where organizations have selected reuse as one of their main strategic weapons, and want to assess and improve their reuse capability. For this purpose the reuse enhancement plan and the reuse maturity model have been defined. By means of these, REBOOT personnel could help a company organize such an initiative. This has also provided valuable experience in motivating an entire organization to adopt reuse.

Component domain	Actual Cost reusable	Estimated Cost non-reusable	KLOC
Time management framework	120	80	3
Simulation framework	60	20	1
Simulation library	100	100	2
Client/server subsystem	80	60	10
Networking library	15 + 200	200	3
Telecom framework	1.5X	X	50

Table 2: Application estimates of extra cost for reusability

As indicated by the list above, a wide variety of applications have taken place, but none of them have used the more advanced technology envisioned in the beginning of the REBOOT project, e.g., the classification, metrics or tools. So far, only minor applications have been conducted with the tool, introducing some reusable classes and frameworks in the REBOOT library. In retrospect, this seems quite natural as most of the application organizations were at a low maturity level when it came to reuse, and those techniques are more appropriate for advanced reuse organizations. It might also be tempting to draw the wider conclusion that reuse is not so much about tool support as many researchers may previously have believed, and that the organizational aspects are far more important, at least at an early stage. Nevertheless, it is believed that the more advanced techniques will be feasible for organizations that have reached a certain reuse maturity level. When a large number of reusable components have been gathered, the need for tool support is likely to show up.

As experienced in most of the applications where products were developed, it costs more to develop a reusable component than a similar non-reusable component — except for cases like the one mentioned above, where a focus on reusability in fact turned out to pay off directly due to changing requirements. Some other estimates from various applications are given in table 2.

The applications are too few in number to make any weighty statements, and the estimated cost for a similar non-reusable artifact is of course somewhat uncertain. It was no surprise that reusable artifacts were more expensive, but the numbers suggest that the overhead cost is usually not unacceptably high: you do not need a large number of reuses to make it pay off. And as indicated above, reduced maintenance costs can make the picture even brighter.

5 Comparison with related work

As mentioned in the introduction, REBOOT's aim has been technology consolidation, and landmark innovations in reuse technology were never the goal. The project has tried to integrate and adapt existing proposals to solve problems with the introduction, organization and technical support for reuse. Hence, it should not be surprising that the REBOOT methodology is highly inspired by previous research.

REBOOT's reuse maturity model, discussed in section 2, is inspired by SEI's CMM [36] for the definition of the maturity levels. Other researchers have also proposed reuse maturity models, e.g., [9, 10, 43]. The selection of key reuse factors for REBOOT's maturity model was most inspired by [43]. All the proposed maturity models can be used for assessing the organization's current maturity level for reuse and suggest a path for stepwise improvement. With the REBOOT model, this stepwise improvement will be the gradual introduction of the REBOOT methodology. Hence, the REBOOT maturity model is a little different in that concrete techniques are provided to fill the improvement slots, whereas the others are more generic. While the maturity model has based much of its content on previous work, the discussion of role conflicts in reuse is a more unique

REBOOT contribution.

The re-engineering method, as discussed in section 3.2, has been inspired by other work in the field, notably [1, 21] on design recovery and [12, 48] on the identification of strongly connected components. Compared to these methods, the REBOOT method is more bottom-up, i.e., more focussed on being used incrementally to harvest some reusable components from a much larger piece of code, rather than re-engineering an entire system. In purpose, it is thus more similar to [34], but this latter approach uses a modularization technique (slicing) which is quite different from REBOOT's. The REBOOT method for identifying strongly connected objects has been made more detailed than its predecessors, to facilitate a higher degree of automation. However, many problems related to re-engineering for reuse remain unsolved. As stated in [19] this is a complex issue. The transition from non-OO to OO design is still a subject for research, although some guidelines for this can be found in [46].

REBOOT's qualification model, discussed in section 3.3, uses the standard factor-criteria-metrics approach also used by IEEE and ISO. Many of the metrics used are well known in the software engineering field, presented in earlier work by, e.g., [5, 14, 23, 26, 31]. However, the *confidence* factor is wholly invented by REBOOT. This factor seems more appropriate for the estimation of reusability than the more well known factor of reliability, because a component which is highly reliable in its original context, may not be so if reused in a different context. Although inspired by other work, REBOOT has made significant extensions for the product metrics, partly based on the answers to questionnaires distributed to experienced European software developers. For instance, several checklists have been developed for metrics which are not obtainable by component parsing.

The approach to classification and retrieval, as discussed in section 3.4, bears strong resemblance to earlier approaches in the field. REBOOT's faceted classification approach is principally the same as that of Prieto-Diaz [37]. However, the exact facets are different. This is because Prieto-Diaz' work was mainly concerned with functional components, whereas REBOOT has been concerned with OO components. For instance, OO components are most naturally abstracted by nouns, functional ones by verbs.

There have been many suggestions for fuzzy retrieval methods in reuse libraries, based on some notion of semantic closeness. REBOOT uses explicitly stated weights connected to the edges in the term-spaces, and is hence most closely related to [4, 16, 35, 37], each of these differing somewhat from each other in the way of structuring the term-spaces and computing distances between request and offer. All these approaches, including REBOOT, have the weakness of being knowledge acquisition intensive. As reported in [44] classification of a number of components and the construction of a term-space requires considerable effort, and the initial estimation of weights will still be ad hoc. Hence, it might be interesting for future improvements of the REBOOT methodology also to investigate other approaches for similarity computation, working automatically on the conceptual structure of the artifact [45] or its documentation [29, 33], so that the initial classification effort can be reduced.

6 Conclusion

6.1 Major contributions of the project

Much of the REBOOT methodology has been highly inspired by previous work, as is normal for a technology consolidation project. Still, some innovations have been made, as mentioned in the previous section. The most important point about REBOOT may be that it has dealt thoroughly both with the technical and organizational aspects of reuse, in a comprehensive manner. Thus, an approach to reuse introduction and organization has been provided *together with* an accompanying technology to fill the various needs an organization will have as its reuse capabilities increase.

Another important contribution of REBOOT is the high emphasis on applications, yielding important practical experience. The most prominent experience made is that organizational aspects can hardly be underestimated. It was found much easier to apply the organizational parts of the methodology than the advanced technology. This was because most application organizations did not have a sufficient reuse maturity to immediately take advantage of the reuse tools.

6.2 Further work

Of course, the methodology may be further improved as more experience is gained, both for the technical and organizational aspects. Many of the applications are continuing after the project itself finished, and the further utilization of these experiences will have to be done by the various partners exploiting the REBOOT results.

For the technical part, one particularly important issue for further work is the integration of the REBOOT environment to more CASE-tools, to make it as widely available as possible. As mentioned, it has already been integrated with the tools CONCERTO and DRAGOON by the partners SEMA and TXT, respectively. Bull also have plans to integrate REBOOT into their CASE offer. Integration with tools other than those provided by the REBOOT partners would of course also be interesting.

For the more advanced parts of the technology it would be interesting to pursue a further elaboration of the classification and retrieval approach, possibly taking up ideas from other ongoing research. Similarly, the metrics part is one of significant scientific interest, and a particularly interesting issue to pursue here would be metrics for the reusability of requirements, since most efforts so far have concentrated on metrics for code and design. More empirical experiments should be a high priority for the advanced parts of the technology, since these turned out to be difficult to use in the applications conducted so far.

Although there are certainly issues that need to be elaborated further, it should be emphasized that the REBOOT methodology is exploitable in its current form, as has already been demonstrated by partners providing consulting services to their customers based on know-how gained through the project and distilled in the Methodology Handbook. Maybe the most important topic for further work will be to keep on with such technology transfer to software producers, thus increasing the general reuse maturity and development productivity, and in turn providing more experience for reuse research.

Acknowledgement

The authors want to acknowledge all the other participants in the REBOOT project. The research reported in this paper could only be realized by the cooperative efforts of many people. Special thanks to the project manager, Jean-Marc Morel, for constructive comments to this paper.

References

- [1] M. Bardieux and P. Delhaise. Step-by-step transition from dusty-deck FORTRAN to object-oriented programming. In *Proc. Reuse, Maintenance and Reverse Engineering of Software '89*, December 1989.
- [2] T. J. Biggerstaff and A. J. Perlis, editors. *Software Reusability, vol. 1: Concepts and Models*. Addison-Wesley, 1989.
- [3] G. Booch. *Object Oriented Design with Applications*. Benjamin/Cummings, 1991.
- [4] J. Börstler. Feature-oriented classification and reuse in IPSEN. Technical report, Aachen University of Technology, 1992.

- [5] T. Bowen et al. Specifications of software quality attributes. Technical report, Griffiss AFB NY 13441-50, February 1985. RADC-TR-85-37, vol.II.
- [6] F. P. Brooks jr. No silver bullet: Essence and accidents of software engineering. In H.-J. Kugler, editor, *Proc. Information Processing '86*. North-Holland, IFIP, 1986.
- [7] B. A. Burton et al. The reusable software library. *IEEE Software*, 4(4):25–33, July 1987. The RSL is developed at Intermedics Inc.
- [8] P. Coad and E. Yourdon. *Object-Oriented Analysis*. Prentice-Hall, Englewood Cliffs, 1990.
- [9] M. Davis. STARS reuse maturity model: guidelines for reuse strategy formulation. In *Proc. 5th Annual Workshop on Software Reuse (WISR'92)*, Palo Alto, California, October 1992.
- [10] T. Davis. The reuse capability model: a basis for improving and organization's reuse capability. In *Proc. 2nd International Workshop on Software Reusability (Reuse'93)*, Lucca, Italy, March 1993.
- [11] A. de Mora and A. Coscuella. A metrics approach to the software reuse problem. In *Proc. 3rd European Conference on Software Quality, Madrid*, November 1992.
- [12] J. Duntemann and C. Marinacci. New objects for old structures. *Byte*, 15(4):261–266, April 1990.
- [13] J. Faget and J.-M. Morel. The REBOOT Environment. In *Proc. 2nd International Workshop on Software Reusability (Reuse'93)*, Lucca, Italy, March 1993.
- [14] N. E. Fenton. *Software Metrics: A Rigorous Approach*. Chapman & Hall, London, 1991.
- [15] W. B. Frakes. Reuse failure modes. In *Proc. 1st International Workshop on Software Reusability, Dortmund*, 1991.
- [16] M. G. Fugini and S. Faustle. Retrieval of reusable components in a development information system. In *Proc. 2nd International Workshop on Software Reusability (Reuse'93)*, Lucca, Italy. IEEE CS Press, March 1993.
- [17] S. Gloss-Soler. The DACS Glossary. A Bibliography of Software Engineering Terms. Technical report, Rome Air Development Center, October 1979.
- [18] HOOD Working Group. HOOD reference manual. Technical Report WME/89-173/JB, European Space Agency, September 1989.
- [19] P. A. V. Hall. Overview of reverse engineering and reuse research. *Information and Software Technology*, 34(4):239–249, April 1992.
- [20] P. A. V. Hall, editor. *Software Reuse and Reverse Engineering in Practice*. Chapman & Hall, 1992.
- [21] I. Jacobson and F. Lindstrom. Re-engineering of old systems to an object-oriented architecture. In *Proc. OOPSLA'91*, 1991.
- [22] R. E. Johnson and V. F. Russo. Reusing object-oriented designs. Technical Report UIUCDCS 91-1696, University of Illinois, May 1991.
- [23] J. M. Juran. *Quality Control Handbook*. McGraw-Hill, New York, 1974.
- [24] E.-A. Karlsson (ed.). *Software Reuse: a holistic approach*. John Wiley & Sons, 1995. REBOOT Methodology Handbook.
- [25] E.-A. Karlsson et al. Weighted term spaces for relaxed search. In *Proc. CIKM'92, Baltimore*. ISMM, November 1992.
- [26] B. Kitchenham. Towards a constructive quality model. Part II. *Software Engineering Journal*, July 1987.
- [27] C. W. Krueger. Software reuse. *ACM Computing Surveys*, 24(2):131–183, June 1992.
- [28] I. Kruzela and M. Brorsson. Human aspects and organizational issues of software reuse. In Hall [20], pages 521–534.
- [29] Y. S. Maarek. An information retrieval approach for building reuse libraries. In *Proc. 1st International Workshop on Software Reusability, Dortmund, Germany*, 1991.
- [30] L. Markosian et al. Using an enabling technology to reengineer legacy systems. *Communications of the ACM*, 37(5):58–70, May 1994.

- [31] J. McCall and M. Matsumoto. Software Quality Assurance, vol.II. Technical report, Griffiss AFB NY 13441-5700, April 1980. RADC-TR-80-109, vol.II.
- [32] H. D. Mills, M. Dyer, and F. C. Linger. Cleanroom software engineering. *IEEE Software*, 4(5):19–25, September 1987.
- [33] J.-Y. Nie, F. Paradis, and J. Vaucher. Using information retrieval for software reuse. In *Computing and Information: Proc. of ICCI'93, Sudbury, Ontario*. IEEE CS Press, 1993.
- [34] J. Q. Ning, A. Engberts, and W. Kozaczynski. Recovering reusable components from legacy systems by program segmentation. In *Proc. Working Conference on Reverse Engineering*. IEEE CS Press, 1993.
- [35] E. Ostertag et al. Computing Similarity in a Reuse Library System: An AI-Based Approach. *ACM Transactions on Software Engineering and Methodology*, 1(3):205–228, July 1992.
- [36] M. C. Paulk, B. Curtis, and M. B. Chrissis. Capability maturity model for software. Technical report, Software Engineering Institute, Pittsburgh, 1991. CMU/SEI-91-TR-24.
- [37] R. Prieto-Diaz and P. Freeman. Classifying software for reusability. *IEEE Software*, 4(1):6–16, January 1987.
- [38] S. R. Ranganathan. *Prolegomena to library classification*. Asian Publishing House, Bombay, 1967.
- [39] M. B. Rosson and J. M. Carroll. A view match for reusing Smalltalk classes. In *Proc. CHI'91*, pages 277–283, 1991.
- [40] S. Shlaer and S. J. Mellor. *Object-Oriented System Analysis: Modeling the World in Data*. Prentice-Hall, Englewood Cliffs, NJ, 1988.
- [41] G. Sindre, E.-A. Karlsson, and P. Paul. Heuristics for maintaining a term space structure for relaxed search. In *Proc. DEXA'92, Valencia*. Springer Verlag, 1992.
- [42] IEEE Computer Society. IEEE Standard Glossary of Software Engineering Terminology. Technical report, IEEE, 1983. ANSI/IEEE Std 729.
- [43] Software Productivity Consortium. Reuse Adoption GuideBook. Technical Report SPC-92051-CMC, version 01.00.03, Software Productivity Consortium Services Corp., November 1992.
- [44] S. Sørungård, G. Sindre, and F. Stokke. Experiences from the application of a faceted classification scheme. In *Proc. 2nd International Workshop on Software Reusability (Reuse'93), Lucca, Italy*. IEEE CS Press, March 1993.
- [45] G. Spanoudakis and P. Constantopoulos. Measuring similarity between software artifacts. In *Proc. SEKE'94, The 6th International Conference on Software Engineering and Knowledge Engineering, Jurmala, Latvia*. Knowledge Systems Institute, 1994.
- [46] P. T. Ward. Object oriented extensions of the Ward-Mellor approach: software development concepts. In *DEC College CASE Methods'92*, 1992.
- [47] R. C. Waters and E. J. Chikofsky (guest editors). Special section on reverse engineering. *Communications of the ACM*, 37(5):22–93, May 1994.
- [48] J. A. Zimmer. Restructuring for style. *Software — Practice and Experience*, 20(4):365–389, April 1990.